PYTHON BASICS

BATCH APRIL 2024

HOUSEKEEPING

- Please keep all questions to the end, we will host a Q&A session at the end. We request everyone to be muted until then.
- Kindly note all questions down or type in the chat box where someone will answer them within the same session.
- We will share recordings of each session in the shared drive within 3 hours of end of session.
- We will be sharing files like ppt, ipnyb(Jupyter notebook) and data files if any before the session in the #data-scientists slack channel , please refer to that for download. We will also publish the same google drive link on the meeting chat once we start.

HISTORY OF PYTHON

Python was designed by Dutch Programmer Guido Van Rossum and released in 1991.

The inspiration for the name of the language comes from a famous BBC comedy show called ‘Monty Python’s Flying Circus’.

Python is older than Java.

Python popularity suddenly grew suddenly from 2010 due to its use in Big Data and Machine learning.

The most used version in Python 3 and the latest version is Python 3.12.3

Why Python ?

Python has always been Open Source. It is one of the top two languages for Data Science besides R.

Simple Syntax and Readability and beginner friendly.

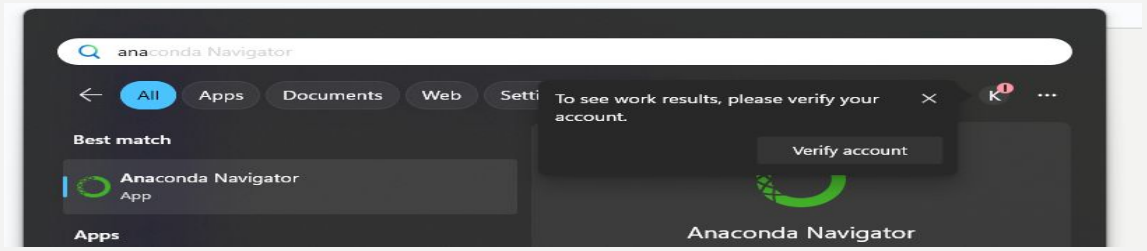
Very easy to build packages in Python especially for Data Science - has already a vast collection of libraries for DS.

It is also used in other domains like web development.

INSTALLATION

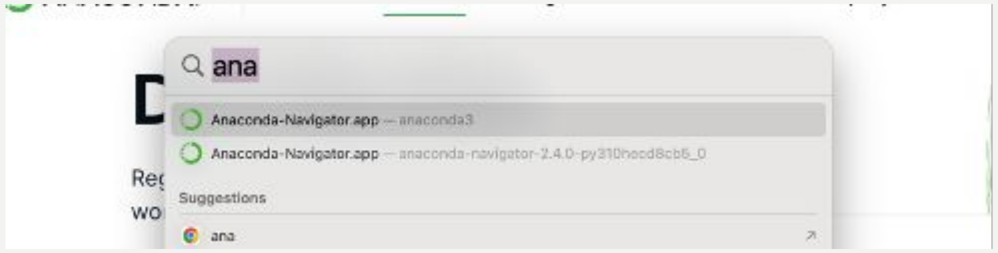
Install Anaconda as per the article
https://www.numpyninja.com/post/step-by-step-installation-instructions-for-anaconda-2024-jupyter-notebook-in-windows
https://docs.anaconda.com/free/anaconda/install/mac-os/

On Windows, go to the
start menu and look for anaconda and start



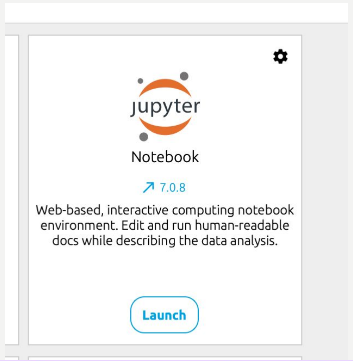
To see work results, please verify your account.

On Mac, access the anaconda app.


ana
Anaconda-Navigator.app — anaconda3
Anaconda-Navigator.app — anaconda-navigator-2.4.0-py310hecd8cb5_0
Suggestions
ana

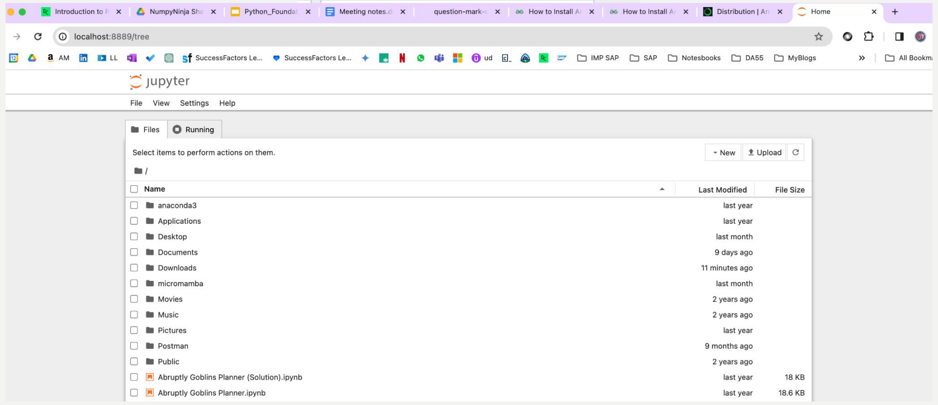
JUPYTER NOTEBOOK

On the Anaconda navigator, click on Jupyter Notebook - Launch.



Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

A local browser will open with the jupyter directory.

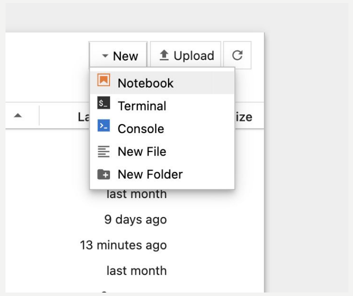


Select items to perform actions on them.

/

Name	Last Modified	File Size
anaconda3	last year	
Applications	last year	
Desktop	last month	
Documents	9 days ago	
Downloads	11 minutes ago	
micromamba	last month	
Movies	2 years ago	
Music	2 years ago	
Pictures	last year	
Postman	9 months ago	
Public	2 years ago	
Abruptly Goblins Planner (Solution).ipynb	last year	18 KB
Abruptly Goblins Planner.ipynb	last year	18.6 KB

Click New > Notebook to create a new notebook file.



Introduction

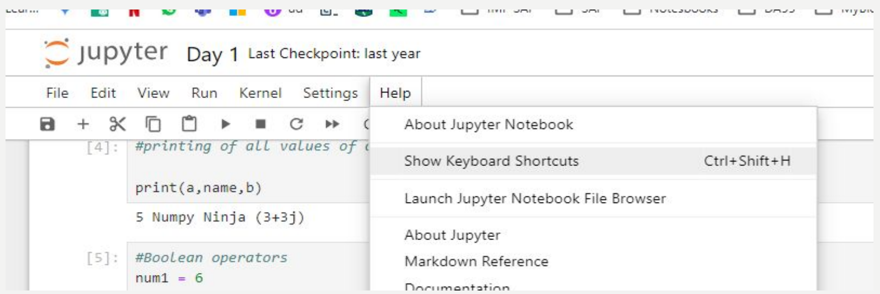
Python code is written in python files with .py extension that can be used in most program editors.

We will be using Jupyter Notebook which generates python notebooks and has extensions **.ipnyb**

Jupyter Notebook uses CLI based approach of executing python code. This means Notebooks can be executed from your terminal using the execute subcommand in each cell.

Jupyter Notebook shortcuts -

Jupyter Notebook>Help> Show Keyboard Shortcuts


Jupyter Day 1 Last Checkpoint: last year
File Edit View Run Kernel Settings Help
About Jupyter Notebook
Show Keyboard Shortcuts Ctrl+Shift+H
Launch Jupyter Notebook File Browser
About Jupyter
Markdown Reference
Documentation

```
[4]: #printing of all values of

     print(a,name,b)
     5 Numpy Ninja (3+3j)

[5]: #Boolean operators
     num1 = 6
```

Most commonly use is **CTRL + ENTER** to execute a cell (replace Command for Control in MAC).

Programming Fundamental concepts I

Python is an Object oriented, **high level programming language (HLL)** -A high-level language is a programming language that uses keywords and statements(Set of instructions) that are similar to expressions in human language or mathematics. It has a readable syntax and needs interpreters to translate the code into machine level language.

Interpreter: An interpreter translates code written in a high-level programming language into machine code line-by-line as the code runs.

Syntax: the set of rules that define the combinations of symbols that make up a correctly structured program. Syntax is important for the compiler/interpreter to understand how to read the code.

Keywords/Reserved words: keywords are predefined words in a programming language with a specific use. One cannot use keywords in any other way than is defined by the syntax of the programming language.

Programming Fundamental concepts II

Datatype : is type of data supported by a programming language. Most HLL support various type of data like integer, real, character, string and boolean.

Variables: a container that stores data values, such as numbers, text, or boolean values. Variables need to be initialized before use by being assigned a value.

Mathematical expressions: A mathematical expression is a statement composed of terms and operations. A term can be a constant, or a variable with a coefficient. An operation can be addition, subtraction, multiplication, or division. An expression may be arithmetic (no variables) or algebraic (with variables).

Loops: is a sequence of instruction is that is continually repeated until a certain condition is reached.

Conditionals: This is a set of instructions which are executed if certain conditions are fulfilled. It also allows for multiple conditions or cases.

Operators: an operator is a character that represents a specific mathematical or logical action or process.

Programming Fundamental concepts III

A programming **library** is a collection of prewritten code that programmers can use to optimize tasks. This collection of reusable code is usually targeted for specific common problems. A library usually includes a few different pre-coded components.

If a block of code needs a specific set of keywords not commonly used or built in, the library needs to be loaded before the remaining code is written.

Functions: A function is a block of organized code that is used to perform a single task and can be recalled by the function name and may have input arguments or necessary data to be fed and returns either a result or performs a task.

A function needs to be declared before it is called and reused which is called invoking the function.

Arguments is basically a set of values passed between programs or functions. It is typically called a parameter and is a way to provide more information to a function.

Programming Fundamental concepts IV

Object: an instance of a particular class or subclass with the class's own methods or procedures and data variables. Object-oriented programming (**OOP**) is a programming type built on objects.

Class: a class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods).

Methods: is equivalent of a function in Object Oriented programming applicable to only objects.

Naming conventions: a naming convention is a set of rules for choosing the character sequence to be used for identifiers which denote variables, types, functions, and other entities in source code and documentation.

Python specifics

No need to declare the type of a variable, python can deduce the type.

There are no sentence enders specially as long as the statements are in different lines.

Indentations are important.

Commenting in Python

Comments are lines in computer programs that are ignored by compilers and interpreters.

Commenting in python simply starts with **#** as a prefix for each line of comment.

Comments can be inline like `1 + 2 # comments here`

Comments help describe what the line of code is meant for making code more readable.

Paragraphs can be commented out with three consecutive double quotes encasing each end.

```
"""
Multiline comment
Here any paragraph can be written
"""
print("comment demo")
```

Popular Keywords/Reserved words/Functions

print: prints the specified message to the screen, or other standard output device.

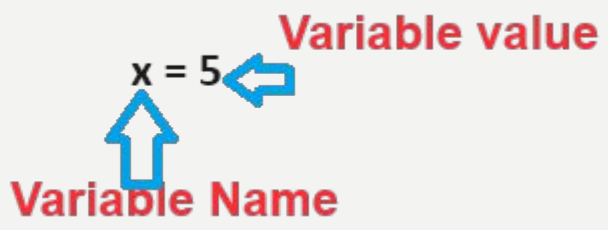
type: a built-in function that is used to return the type of data stored in the objects or variables in the program.

input: allows user input. *Syntax*. input(prompt).

Demo - Getting started

VARIABLES

Syntax:



Python Variable is a container that stores value.

Specific, case- sensitive name of any value

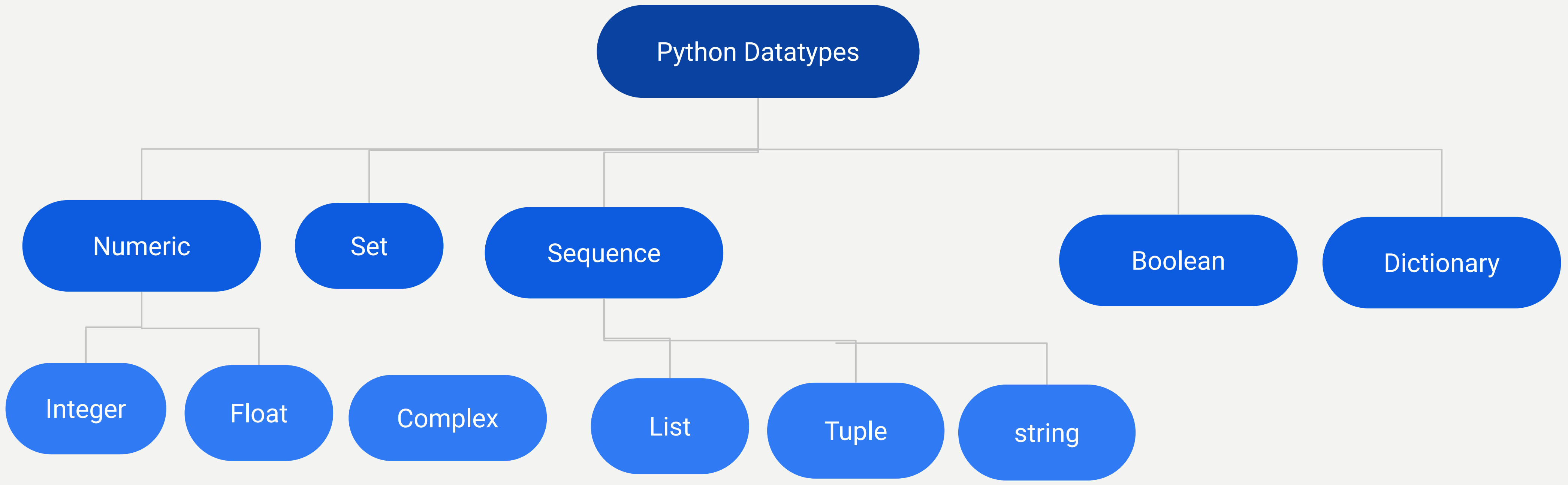
Does not need any qualifying of data type

Assigned using assignment operator ‘=’.

The left side is always the variable and the right side is always the value.

Naming Convention: variable names should all be lower case
Space is not accepted in a variable where an underscore can be used.

DATA TYPES


Python Datatypes
Numeric
Set
Sequence
Boolean
Dictionary
Integer
Float
Complex
List
Tuple
string

Numeric Data types

There are three distinct numeric types: integers, floating point numbers, and complex numbers.

Integer: These are whole numbers which are positive or negative and represented by int.

There is no limit to how long an integer can be in python.

Float: Float numbers usually contain a decimal point and even whole numbers are represented ending with .0. This helps representing fractional numbers.

Complex: represented with the numeric complex numbers specified as (real part) + (imaginary part)j where j is square root of -1 (imaginary number).For example – 2+3j

Sequence Data Types

Sequence data type is an ordered set in Python and there are several types of sequences, some of them are more used than others.

List: Lists are an array of data and is mutable meaning they can be changed. It is created by placing the data inside the square brackets [] or **list()** function and separating all values by comma. Data type inside the list may not be the same. Since they are ordered, they have indices.

Tuples: Similar to lists but are immutable, they cannot be changed after creation. They can be created using **tuple()** function or parenthesis ().

Strings: this is sequence of character data and represented by str or put between single or double quotes.

Other Data types

Boolean: is the datatype that supports True and False values. Each value in sentence case is a reserved keyword.

Set: a set is an unordered collection data type that is iterable, mutable, and has no duplicate elements. In python sets are written with curly brackets.

Note: sets are unordered, so the items will appear in a random order.

- Set values will be embedded in {}
- Set will allow only the unique values (in case if there are any duplicates then those duplicate value should be removed)
- Insertion order is not maintained.
- Set will not support indexing

Dictionary: - each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Demo - Variables & Data Types

jupyter

OPERATORS I

1.**Assignment operator** - python assignment operators are used for assigning the value of the right operand to the left operand. Various assignment operators used in python are **(=, +=, - =, *=, /=,** etc.)

2.**Relational operator** - relational operators are used to establish some sort of relationship between the two operands. Python language is capable of understanding these types of operators and accordingly return the output, which can be either true or false (**>, <,>=, <=,! =,== etc.)**

3.**Logical operator** - logical operators, as the name suggests are used in logical expressions where the operands are either true or false. The operands in a logical expression can be expressions which return true or false upon evaluation. There are three basic types of logical operators:

•**and (&)**
•**or (|)**
•**not**

OPERATORS II

4.Arithmetic operator - arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc. (**+, - ,*, /, %, //, ****Etc.)

5.Membership operator - IN , NOT IN

They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

•In a dictionary we can only test for presence of key, not the value.

6.Identity operator -IS and IS NOT

They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

Source: https://www.w3schools.com/python/python_operators.asp

Demo - Operators

LISTS I

The basic collection of data in Python which is one of the built in data structure in a sequential order.

Empty list is allowed i.e. null values with []

List allows duplicate elements

List is mutable and maintains a sequential order of insertion

List is indexed

LISTS II

Method	Description
append()	Used for adding elements to the end of the List.
copy()	It returns a shallow copy of a list
clear()	This method is used for removing all items from the list.
count()	These methods count the elements.
extend()	Adds each element of an iterable to the end of the List
index()	Returns the lowest index where the element appears.
insert()	Inserts a given element at a given index in a list.
pop()	Removes and returns the last value from the List or the given index value.
remove()	Removes a given object from the List.
reverse()	Reverses objects of the List in place.
sort()	Sort a List in ascending, descending, or user-defined order
min()	Calculates the minimum of all the elements of the List
max()	Calculates the maximum of all the elements of the List

Demo - Lists

TUPLES I

A tuple can be used for storing a group of comma separated elements enclosed within ()

Tuple allows duplicate Elements

Tuple allows null values

Tuple will maintains the insertion Order

Once created, a tuple is immutable or unchangeable

Tuple is indexed

TUPLES II

Method	Description
count()	returns occurrences of element in a tuple
index()	returns smallest index of element in tuple
len()	Returns Length of an Object
max()	returns largest element
min()	returns smallest element
sorted()	returns sorted list from a given iterable
sum()	Add items of an Iterable
tuple()	Creates a Tuple

SET I

A set can be used for storing a group of elements enclosed within {} and is not order based.

A set does not allow duplicate Elements.

A set allows null values.

A set is mutable.

SET II

Method	Description
remove()	Removes Element from the Set
add()	adds element to a set
copy()	Returns Shallow Copy of a Set
clear()	remove all elements from a set
difference()	Returns Difference of Two Sets
discard()	Removes an Element from The Set
intersection()	Returns Intersection of Two or More Sets
issubset()	Checks if a Set is Subset of Another Set
issuperset()	Checks if a Set is Superset of Another Set
pop()	Removes an Arbitrary Element
symmetric_difference()	Returns Symmetric
Difference union()	Returns Union of Sets
update()	Add Elements to The Set
len()	Returns Length of an Object
max()	returns largest element

Demo - Tuple & Set

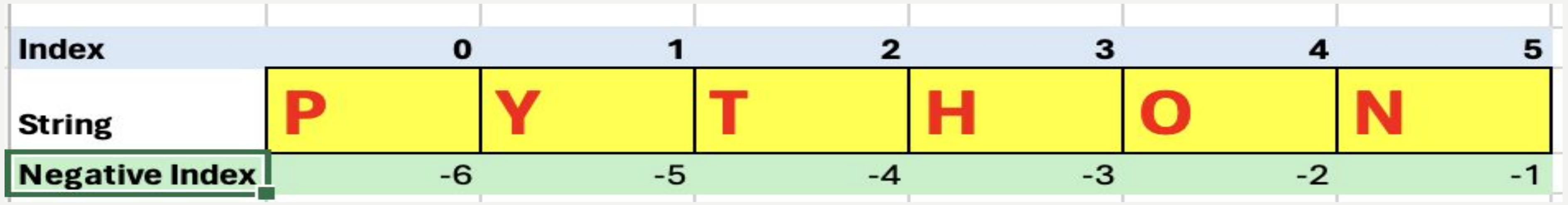
STRINGS I

A string is a sequence of characters contained within a pair of 'single quotes' or "double quotes".

A string can be any length and can contain any letters, numbers, symbols, and spaces.

A string can be thought of as a list of characters.

STRINGS II



Index	0	1	2	3	4	5
String	P	Y	T	H	O	N
Negative Index	-6	-5	-4	-3	-2	-1

Like any other list, each character in a string has an index. Consider the above word.

The index of the characters are given above the word while the reverse index is given below the word.

A string can be sliced with the help of indices. The slice will contain a part of the string and this process is called slicing.

Both positive and negative indices can be used for slicing a string. This includes any type of characters including commas, characters and spaces.

STRINGS III

Methods	Description
capitalize()	Converts the first character of the string to a capital (uppercase) letter
casefold()	Implements caseless string matching
center()	Pad the string with the specified character.
count()	Returns the number of occurrences of a substring in the string.
encode()	Encodes strings with the specified encoded scheme
endswith()	Returns “True” if a string ends with the given suffix
expandtabs()	Specifies the amount of space to be substituted with the “\t” symbol in the string
find()	Returns the lowest index of the substring if it is found
format()	Formats the string for printing it to console
format_map()	Formats specified values in a string using a dictionary
index()	Returns the position of the first occurrence of a substring in a string
isalnum()	Checks whether all the characters in a given string is alphanumeric or not
isalpha()	Returns “True” if all characters in the string are alphabets
isdecimal()	Returns true if all characters in a string are decimal
isdigit()	Returns “True” if all characters in the string are digits
isidentifier()	Check whether a string is a valid identifier or not
islower()	Checks if all characters in the string are lowercase
isnumeric()	Returns “True” if all characters in the string are numeric characters
isprintable()	Returns “True” if all characters in the string are printable or the string is empty

STRINGS IV

Methods	Description
swapcase()	Converts all uppercase characters to lowercase and vice versa
title()	Convert string to title case
translate()	Modify string according to given translation mappings
upper()	Converts all lowercase characters in a string into uppercase
zfill()	Returns a copy of the string with ‘0’ characters padded to the left side of the string

Methods	Description
isspace()	Returns “True” if all characters in the string are whitespace characters
istitle()	Returns “True” if the string is a title cased string
isupper()	Checks if all characters in the string are uppercase
join()	Returns a concatenated String
ljust()	Left aligns the string according to the width specified
lower()	Converts all uppercase characters in a string into lowercase
lstrip()	Returns the string with leading characters removed
maketrans()	Returns a translation table
partition()	Splits the string at the first occurrence of the separator
replace()	Replaces all occurrences of a substring with another substring
rfind()	Returns the highest index of the substring
rindex()	Returns the highest index of the substring inside the string
rjust()	Right aligns the string according to the width specified
rpartition()	Split the given string into three parts
rsplit()	Split the string from the right by the specified separator
rstrip()	Removes trailing characters
splitlines()	Split the lines at line boundaries
startswith()	Returns “True” if a string starts with the given prefix
strip()	Returns the string with both leading and trailing characters

Demo - Strings

Q & A

Previously, we learned how we can use lists in Python to store multiple values, or elements, in a single variable. Each element in a list is given an index, which allows us to uniquely reference and identify each element's position in a list. Python lists use consecutive integers starting at
 as indexes, which makes for a pretty easy to use data structure.

However, what if we would like to use some other value besides an integer as the index? For example, if we create a data structure to store information about a user account, wouldn't it be much easier if we could use the username as the index, allowing us to quickly find that particular user's data?

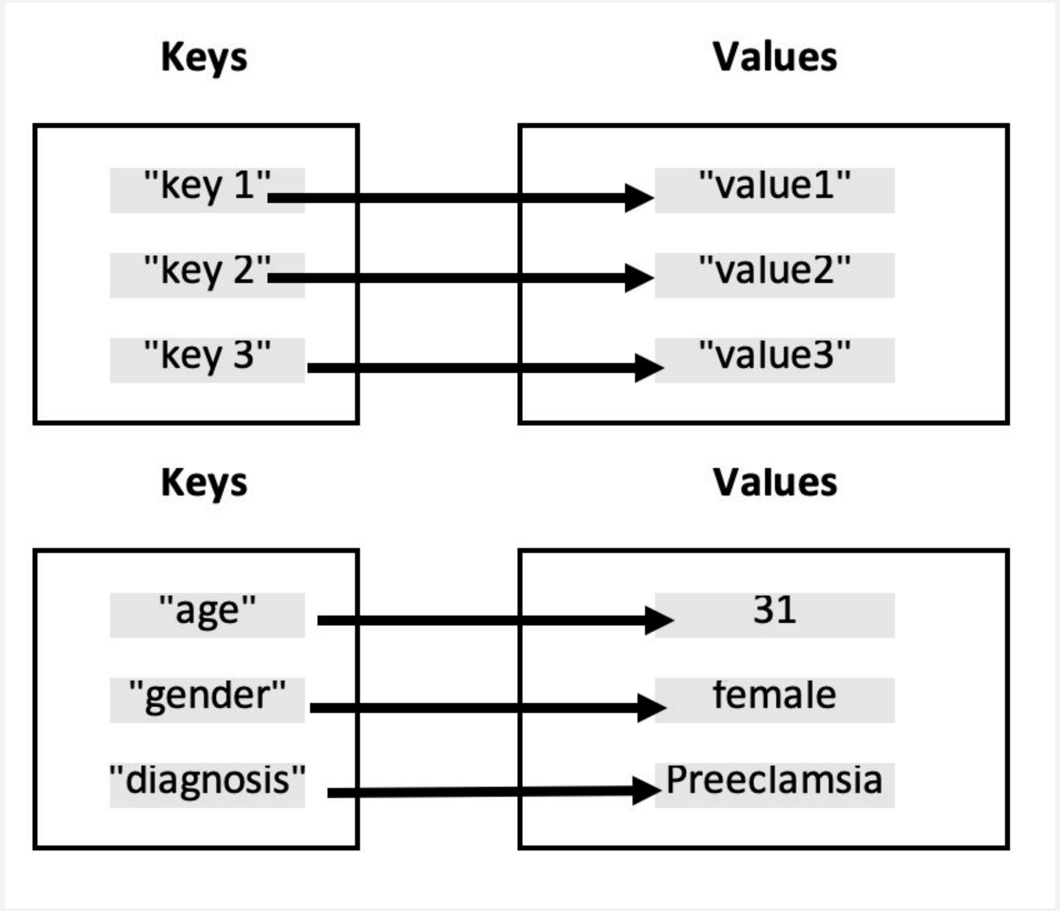
Thankfully, Python includes another built-in data structure, called a dictionary, that allows us to do exactly that. In a dictionary, instead of indexes we have keys, which can be any data type, that uniquely identify each item, or value, in the dictionary. Other programming languages refer to dictionaries as hash tables, hash maps, or associative arrays, which gives a clue to how they work behind the scenes.

DATA TYPE - DICTIONARY {}

- Python dictionary is another built in data structure to store data in key: value pairs. Each key-value pair maps the key to its associated value.
- Dictionary is enclosed within {}

Dictionary rules

- keys are immutable (cannot be changed). Key can not be a mutable data type, for example, a list.
- data types that can be either strings or numbers.
- Keys are unique within a dictionary and can not be duplicated inside a dictionary, in case if it is used more than once then subsequent entries will overwrite the previous value.



DICTIONARY GUIDE (1/2)

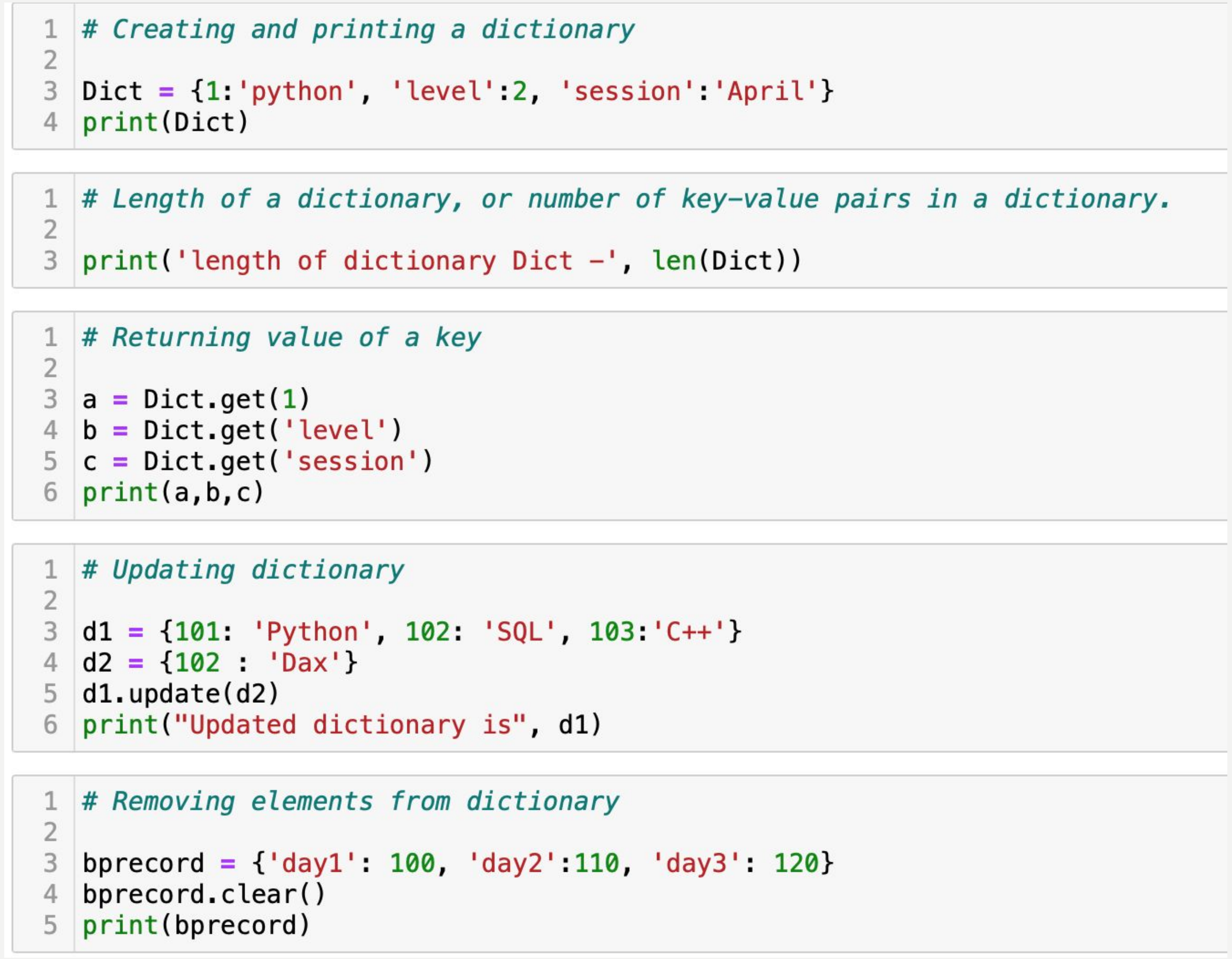
Method	Description
clear()	Removes all Items
del	Delete dictionary using 'del' keyword.
copy()	Returns Shallow Copy of a Dictionary
get()	Returns Value of The Key
items()	Returns view of dictionary's (key, value) pair
keys()	Returns View Object of All Keys
popitem()	Returns & Removes Element From Dictionary
setdefault()	Inserts Key With a Value if Key is not Present
pop()	Removes and returns element having given key
values()	Returns view of all values in dictionary
update()	Updates the Dictionary
iter()	Get a key iterator for a dictionary
in/not in	Check if a key is present in a dictionary using 'in'.
dict()	Creates a Dictionary
len()	Returns Length of an Object
max()	returns largest element
min()	returns smallest element
map()	Applies Function and Returns a List
sorted()	Returns sorted list from a given iterable
sum()	Add items of an Iterable

DICTIONARY GUIDE (2/2)

```
dict_1 = {}
```

```
dict_2 = {"a": 1, "b": 2,
"c": 3}
```

```
dict_3 = {
   "cat": "mammal",
   "lizard": "reptile",
   "goldfish": "fish",
   "chickadee": "bird"
}
```



```python
# Creating and printing a dictionary

Dict = {1:'python', 'level':2, 'session':'April'}
print(Dict)
```

```python
# Length of a dictionary, or number of key-value pairs in a dictionary.

print('length of dictionary Dict -', len(Dict))
```

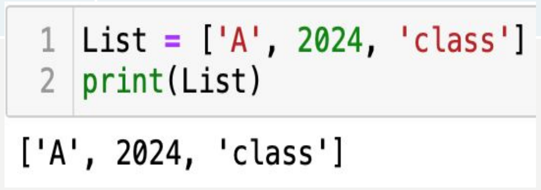
```python
# Returning value of a key

a = Dict.get(1)
b = Dict.get('level')
c = Dict.get('session')
print(a,b,c)
```

```python
# Updating dictionary

d1 = {101: 'Python', 102: 'SQL', 103:'C++'}
d2 = {102 : 'Dax'}
d1.update(d2)
print("Updated dictionary is", d1)
```

```python
# Removing elements from dictionary

bprecord = {'day1': 100, 'day2':110, 'day3': 120}
bprecord.clear()
print(bprecord)
```

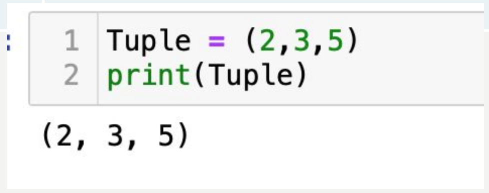
COMPARISON OF THE IN-BUILT DATA STRUCTURES

	List	Tuple	Set	Dictionary
	[]	()	{}	{}
Duplicates	Allowed	Allowed	Not Allowed	Key - not allowed
				Value - allowed
Insertion order	Ordered	Ordered	Unordered	Ordered
Null	Allowed	Allowed	Allowed	Key - not allowed
				Value - allowed
Data retrieve	Index	Index	No Index	Key
Mutability	Mutable	Immutable	Mutable	Key - immutable,
				Value - mutable
Slicing	Can be done	Can be done	not done	Not done
Syntax				



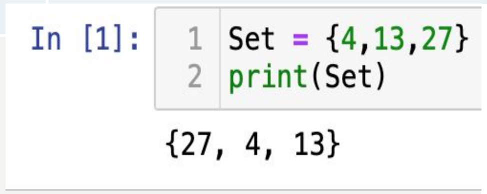
```
List = ['A', 2024, 'class']
print(List)
```

```
['A', 2024, 'class']
```



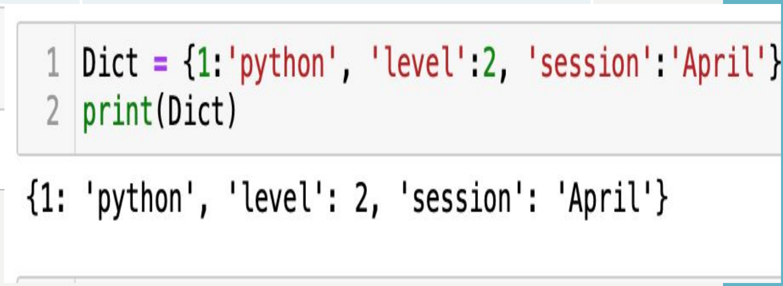
```
Tuple = (2,3,5)
print(Tuple)
```

```
(2, 3, 5)
```



```
In [1]: Set = {4,13,27}
        print(Set)
```

```
{27, 4, 13}
```

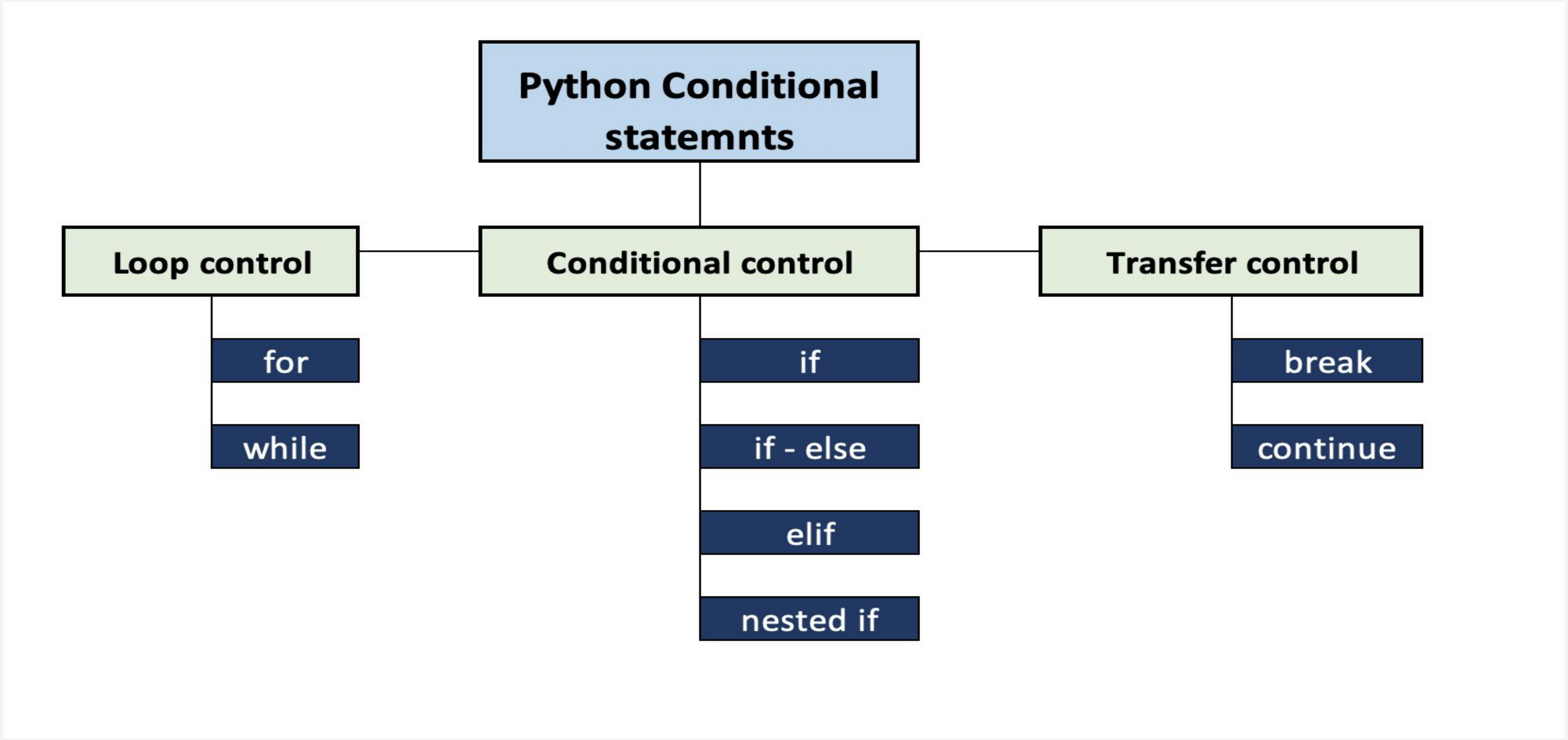


```
Dict = {1:'python', 'level':2, 'session':'April'}
print(Dict)
```

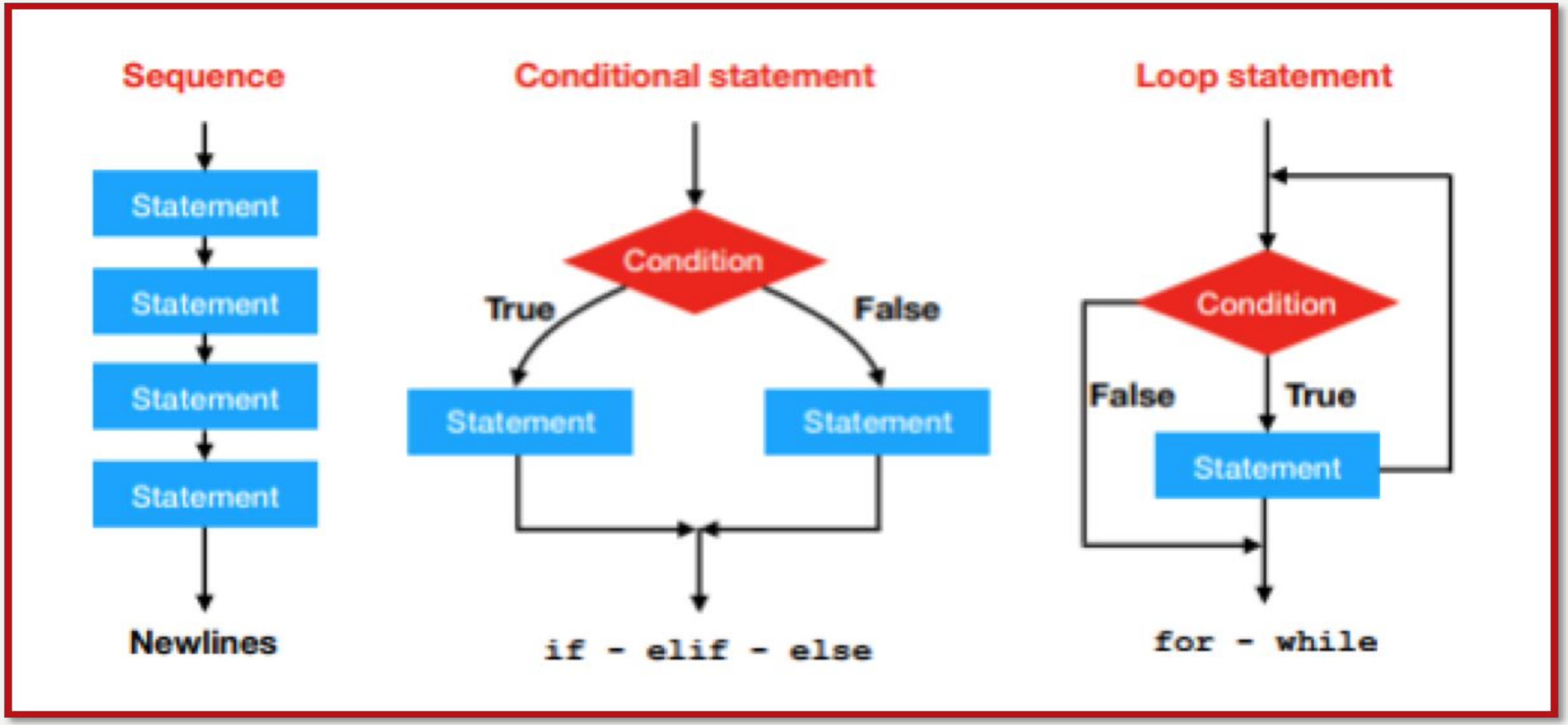
```
{1: 'python', 'level': 2, 'session': 'April'}
```

CONTROL FLOW IN PYTHON

CONTROL FLOW STATEMENT



CONTROL FLOW STATEMENT STRUCTURES



When a program contains a conditional statement, its control flow can split into different branches. This means that each program execution may take a different path through the program, but it will eventually reach the end. There is no way to go back and repeat a step that has already been done!

CONDITIONAL STATEMENT - IF

Decision-making helps in deciding the flow of execution of the program It simply decides whether a particular code block will be executed or not on the basis of the condition provided in the if statement. If the condition true, then the code block is executed, and if it's false then the code block is not executed.

If statement:

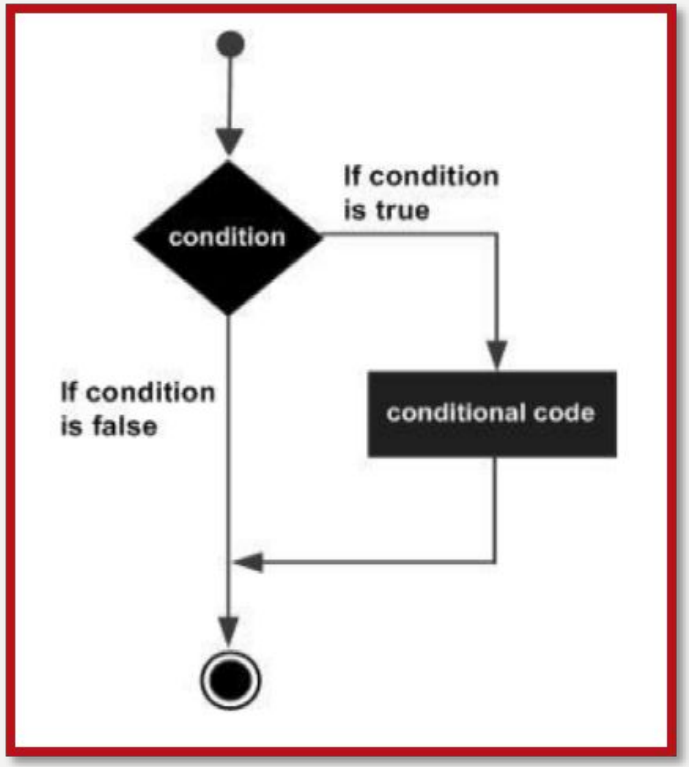
used to execute a group of statements based on the result of a condition.

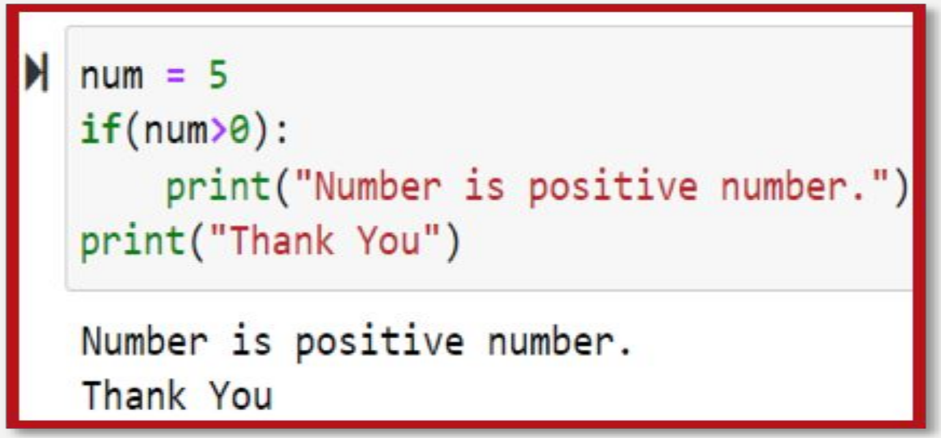
Syntax:

If(codition):

    Statement-1

    Statement-2





```
num = 5
if(num>0):
    print("Number is positive number.")
print("Thank You")
```

```
Number is positive number.
Thank You
```

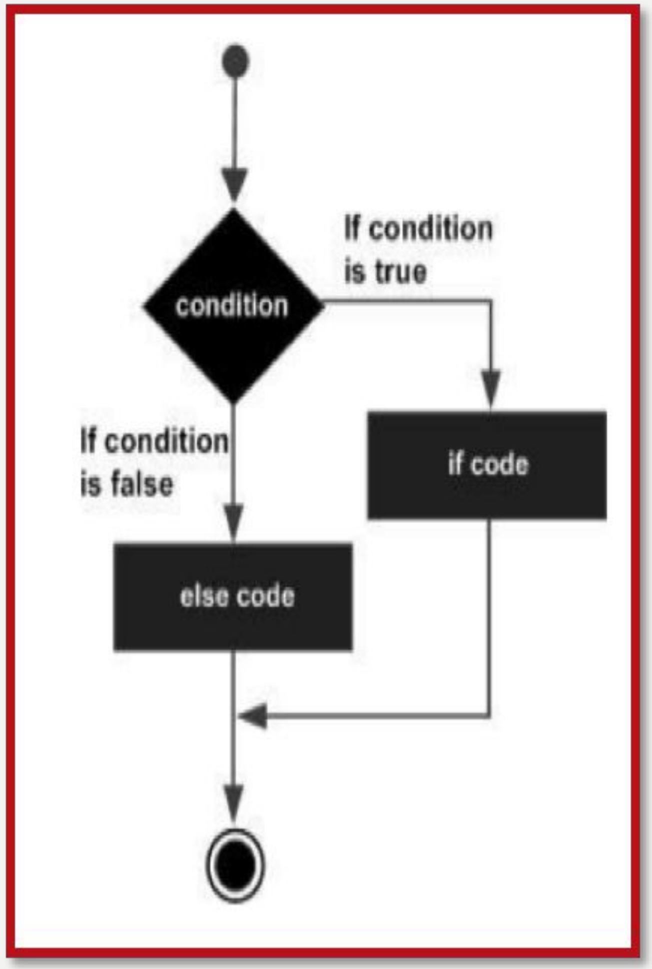
CONDITIONAL STATEMENT - IF ELSE

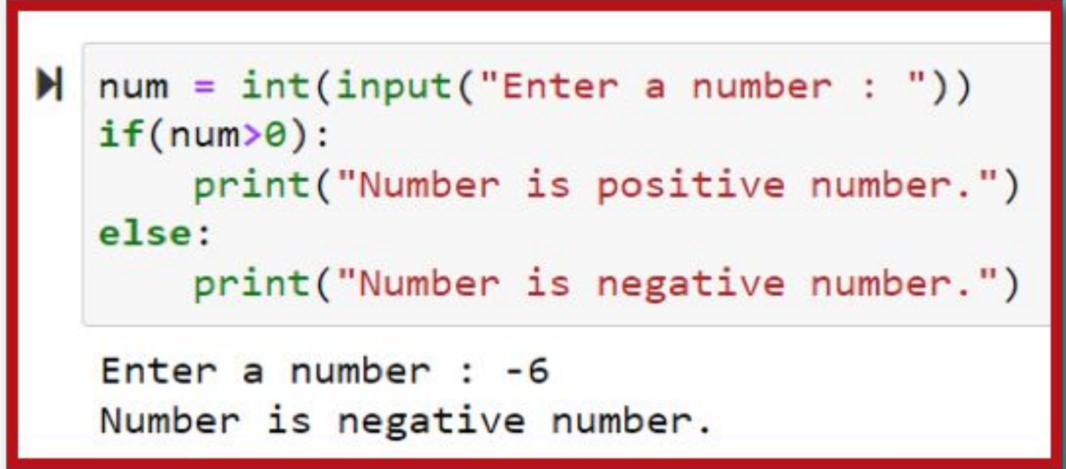
If else:

used to execute group of statements based on a condition, if the condition is true then it will execute if-block otherwise it will execute else-block

Syntax:

```
if(condition):
    Statement-1
else:
    Statement-2
```





```
num = int(input("Enter a number : "))
if(num>0):
    print("Number is positive number.")
else:
    print("Number is negative number.")
```

```
Enter a number : -6
Number is negative number.
```

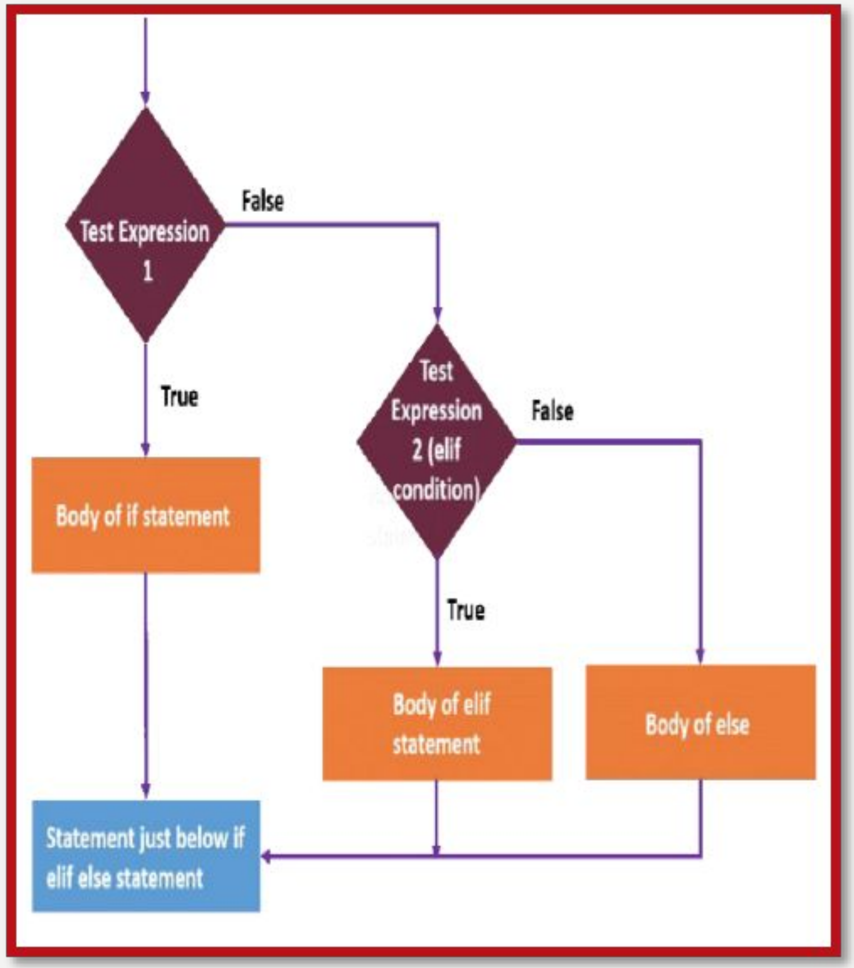
CONDITIONAL STATEMENT - ELIF

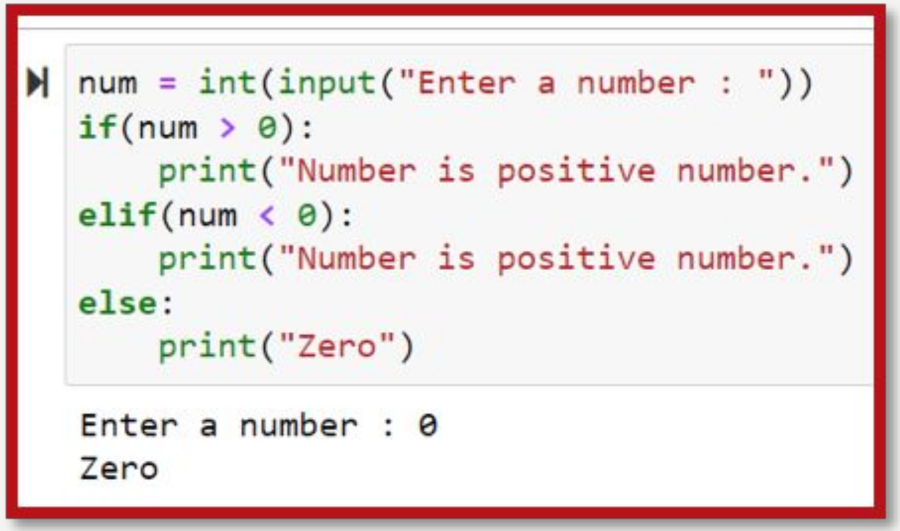
Elif :

is used when you need to choose one between more than two alternatives.

Syntax:

```
if(condition):
      Statement -1
elif(condition):
      Statement -2
else:
      Statement -3
```





```
num = int(input("Enter a number : "))
if(num > 0):
    print("Number is positive number.")
elif(num < 0):
    print("Number is positive number.")
else:
    print("Zero")
```

```
Enter a number : 0
Zero
```

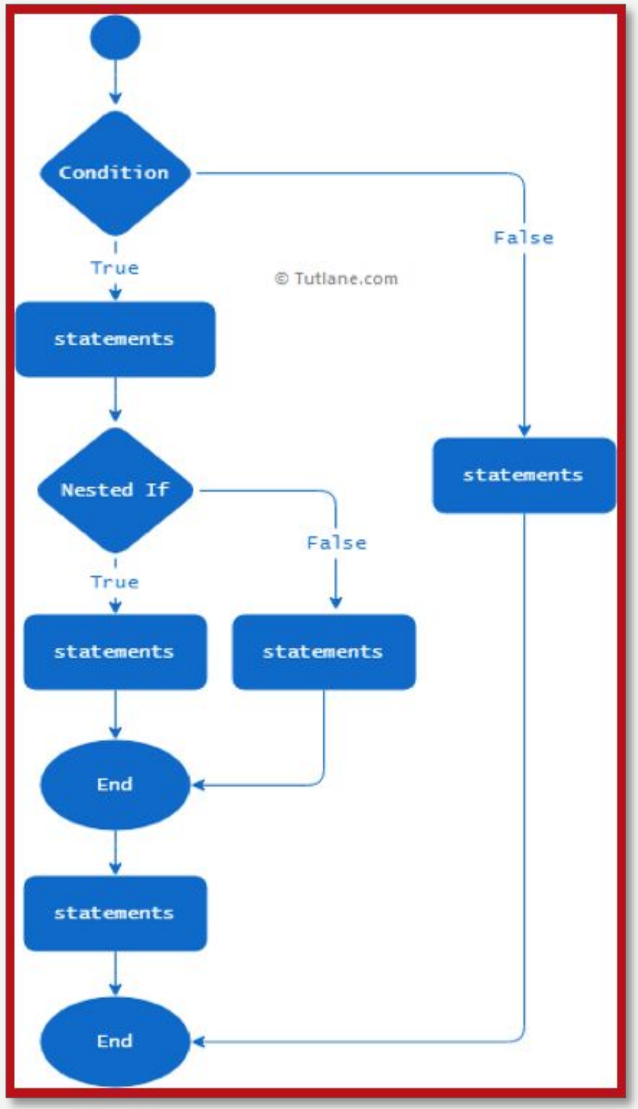
CONDITIONAL STATEMENT - NESTED IF

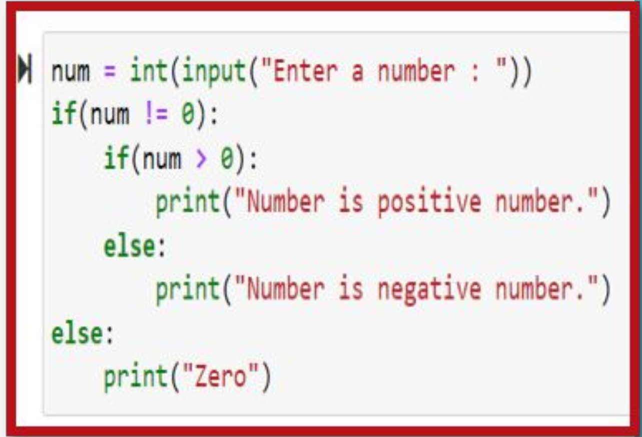
Nested if:

If Statement presents inside of another if statement then it is called as nested-if.

Syntax

```
if expression1:
   statement(s)
   if expression2:
      statement(s)
   elif expression3:
      statement(s)
   elif expression4:
      statement(s)
   else:
      statement(s)
else:
   statement(s)
```





```
num = int(input("Enter a number : "))
if(num != 0):
    if(num > 0):
        print("Number is positive number.")
    else:
        print("Number is negative number.")
else:
    print("Zero")
```

LOOPING / ITERATIVE STATEMENT Intro

Range function
- range(stop_number) - with a single argument, the range() function will generate a list of numbers
- the range will begin at 0 and stops before it reaches the stop_number value.

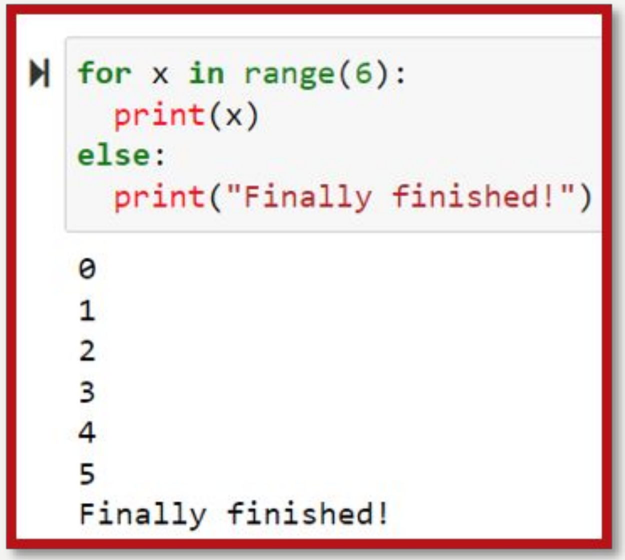
- For example, range(10) will generate a list of numbers from from 0 to 9.
- range(start, stop, step)- with three arguements will generate a list of numbers as follows -
- Begins at start, increases by step each time, and stops before the stop value.

list(range(5,15))

[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
In [20]:

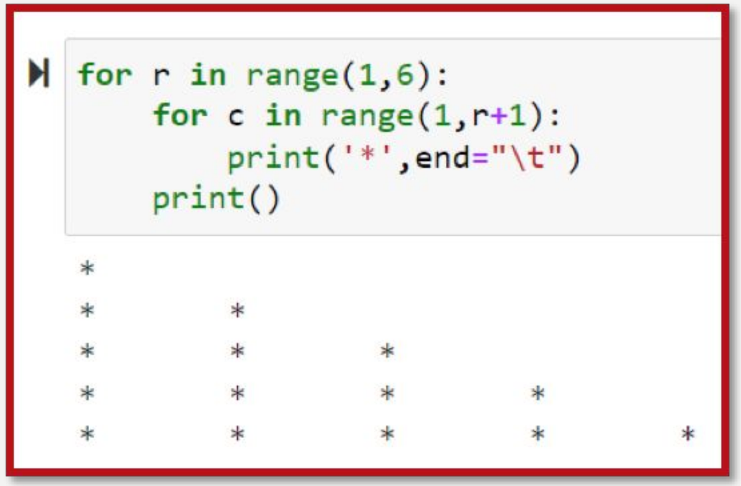
list(range (0,10,2))

[0, 2, 4, 6, 8]



```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
3
4
5
Finally finished!
```



```
for r in range(1,6):
    for c in range(1,r+1):
        print('*',end="\t")
    print()
```

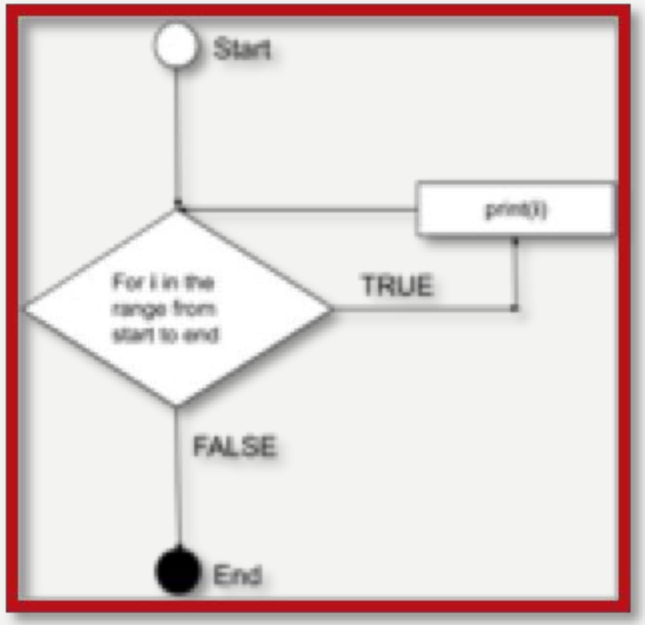
```
*
*	*
*	*	*
*	*	*	*
*	*	*	*	*
```

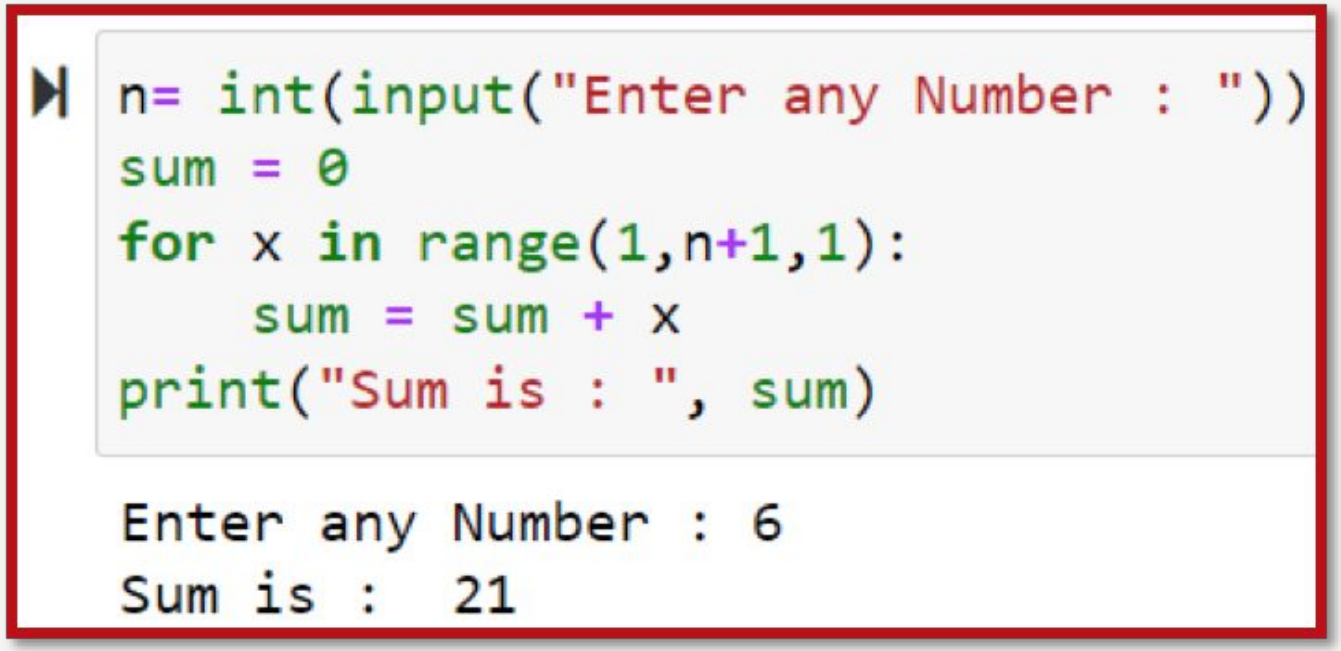
LOOPING / ITERATIVE STATEMENT for

for loop:
"for loop" is a repetition process which is used to execute group of statements repeatedly within the given range.

Syntax:

```
for variable in range()/str/sequence:
        statement1
        statement2
.......
```





```
n= int(input("Enter any Number : "))
sum = 0
for x in range(1,n+1,1):
    sum = sum + x
print("Sum is : ", sum)
```

```
Enter any Number : 6
Sum is :  21
```

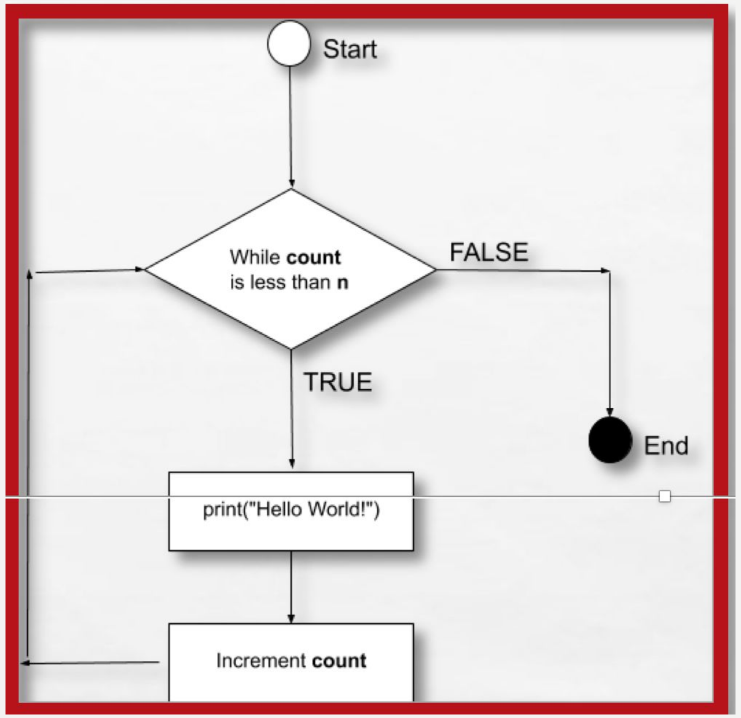
LOOPING / ITERATIVE STATEMENT - while

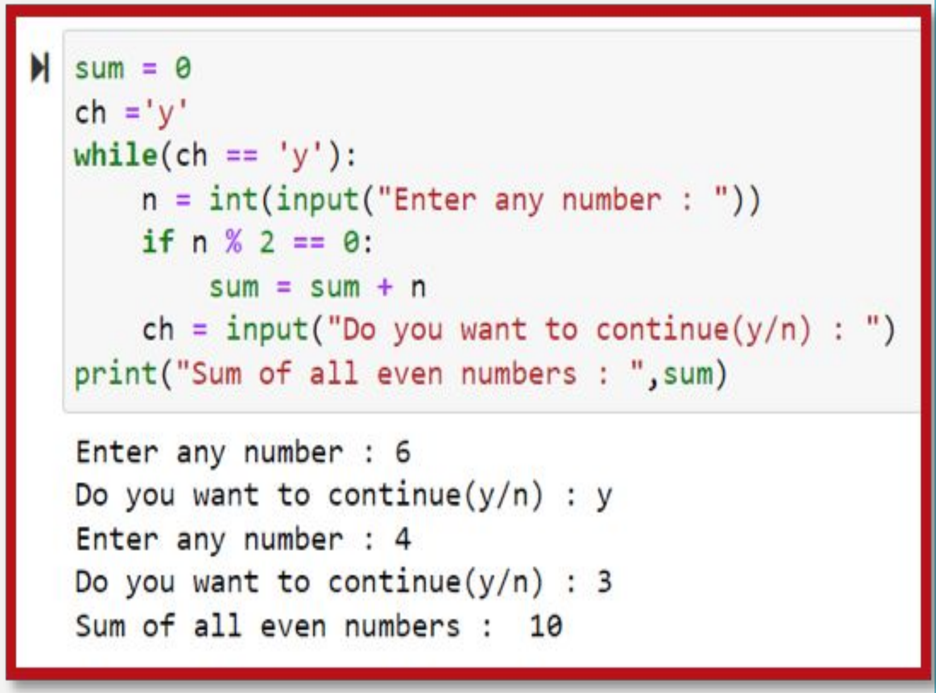
while loop:

"while loop" is executing group of statements repeatedly based on the condition as long as a condition is true.

Syntax:

```
while <condition>:
    statement(s)
```





```
sum = 0
ch ='y'
while(ch == 'y'):
    n = int(input("Enter any number : "))
    if n % 2 == 0:
        sum = sum + n
    ch = input("Do you want to continue(y/n) : ")
print("Sum of all even numbers : ",sum)
```

```
Enter any number : 6
Do you want to continue(y/n) : y
Enter any number : 4
Do you want to continue(y/n) : 3
Sum of all even numbers :  10
```

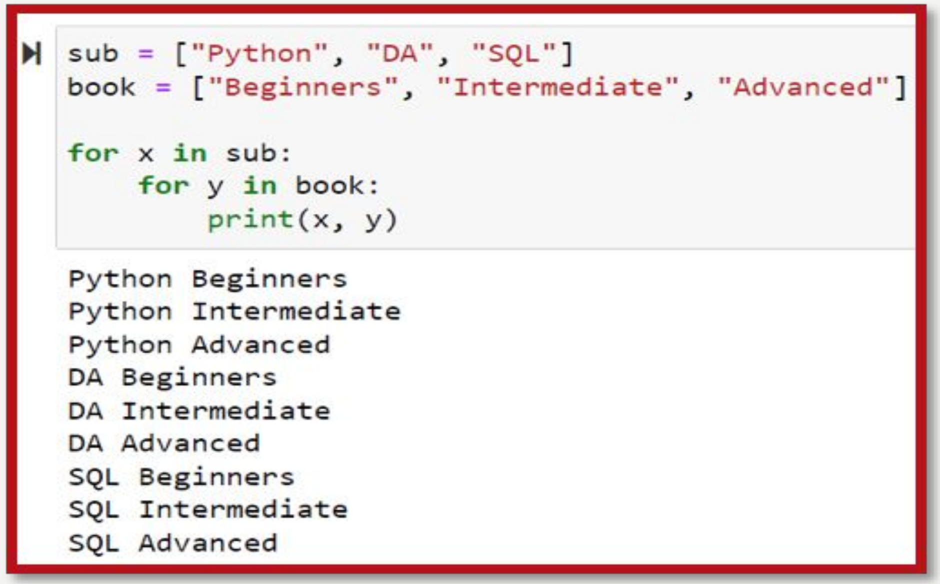
LOOPING / ITERATIVE STATEMENT - NESTED for

1. Nesting For Loops

for loops can be nested within themselves.

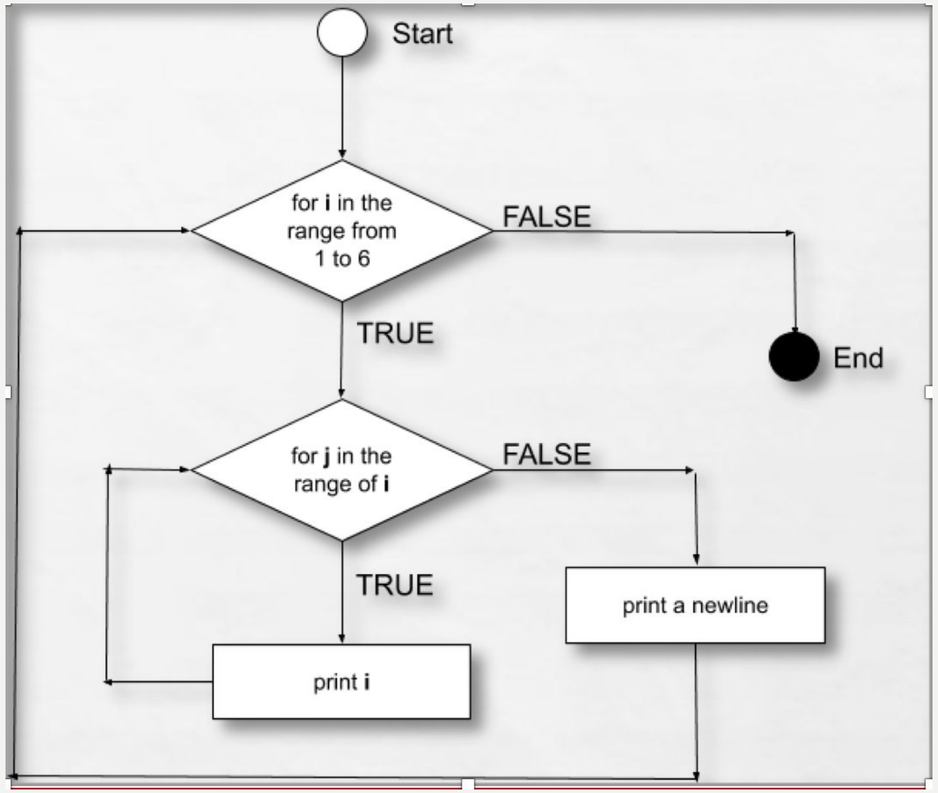
The **syntax** below shows a 1-level nested for loop.

```
for in n:
    # piece of code goes here
    for in n:
        #  piece of code goes here
```



```
sub = ["Python", "DA", "SQL"]
book = ["Beginners", "Intermediate", "Advanced"]

for x in sub:
    for y in book:
        print(x, y)
```

```
Python Beginners
Python Intermediate
Python Advanced
DA Beginners
DA Intermediate
DA Advanced
SQL Beginners
SQL Intermediate
SQL Advanced
```



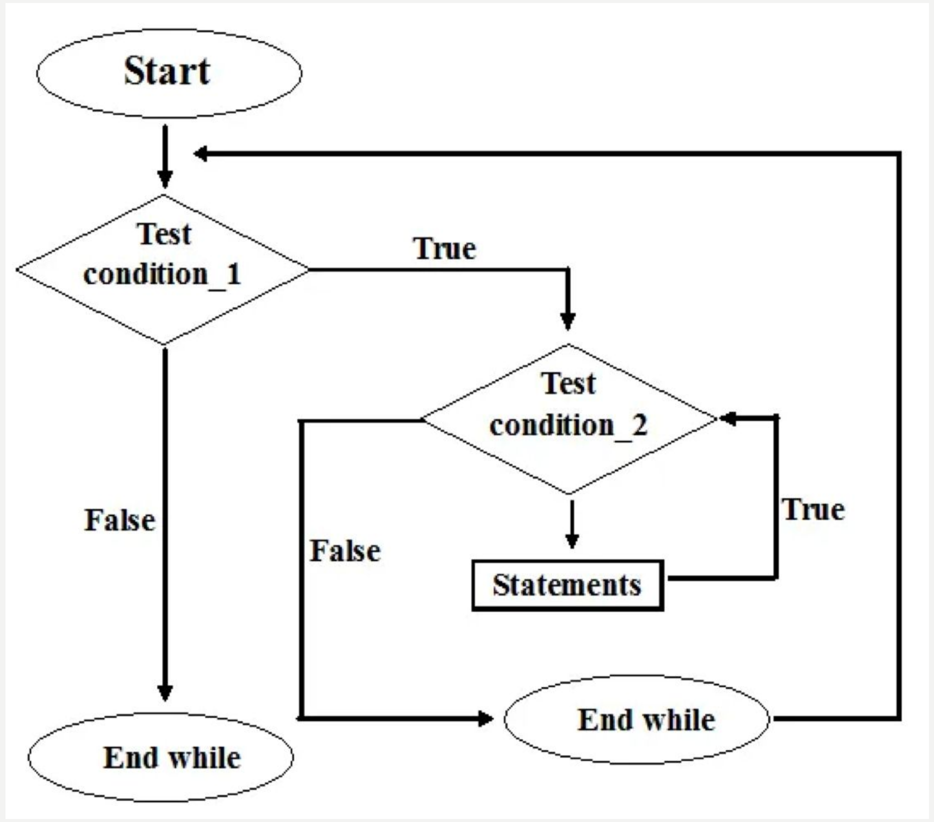
LOOPING / ITERATIVE STATEMENT - NESTED while

2. Nesting While Loops

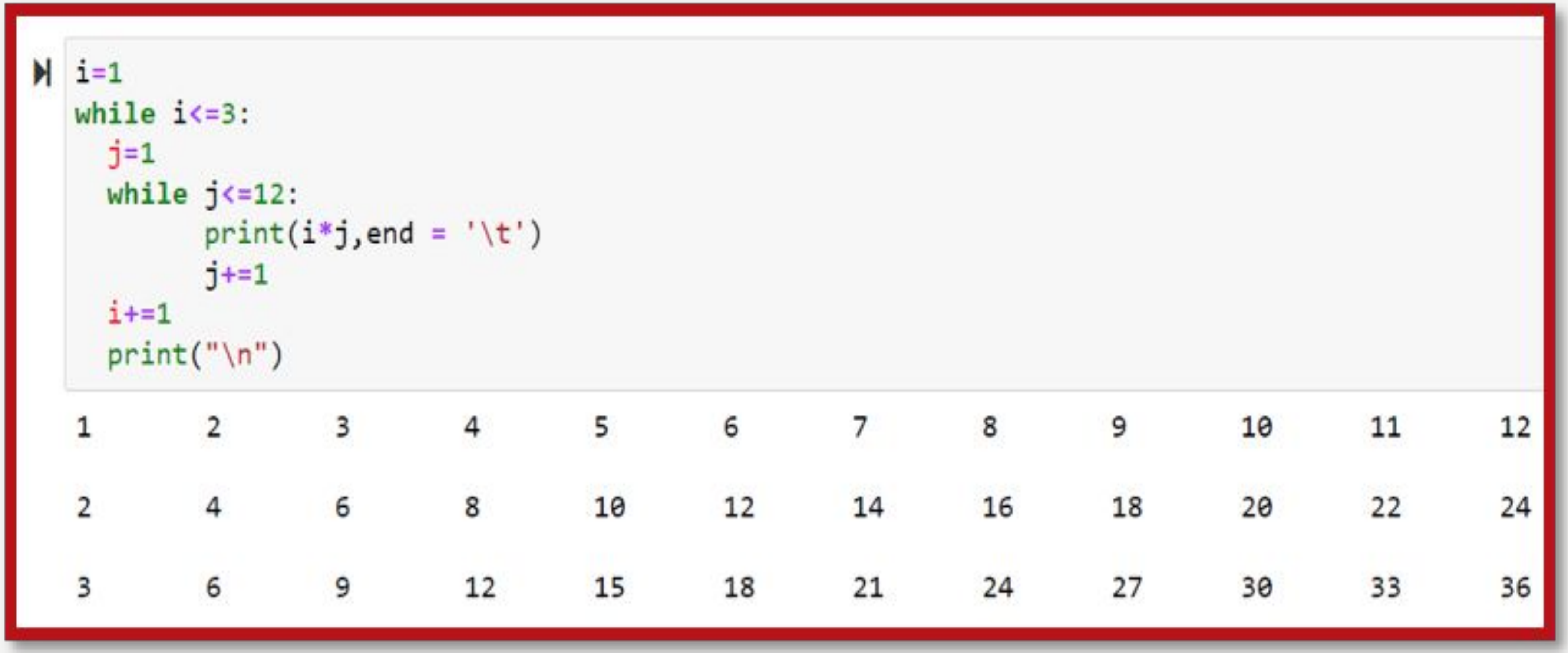
While loops can be nested within themselves.

The syntax below shows a 1-level nested while loop.

```
while condition:
    # piece of code goes here
    while condition:
        #  piece of code goes here
```



LOOPING / ITERATIVE STATEMENT - NESTED while



```
i=1
while i<=3:
  j=1
  while j<=12:
        print(i*j,end = '\t')
        j+=1
  i+=1
  print("\n")
```

```
1	2	3	4	5	6	7	8	9	10	11	12

2	4	6	8	10	12	14	16	18	20	22	24

3	6	9	12	15	18	21	24	27	30	33	36
```

TRANSFER STATEMENTS

1. Continue Statement:

When the program encounters a continue statement, it will skip the statements which are present after the continue statement inside the loop and proceed with the next iterations.

Syntax:

continue

2. Break Statement:

The break statement is used to terminate the loop containing it, the control of the program will come out of that loop.

Syntax:

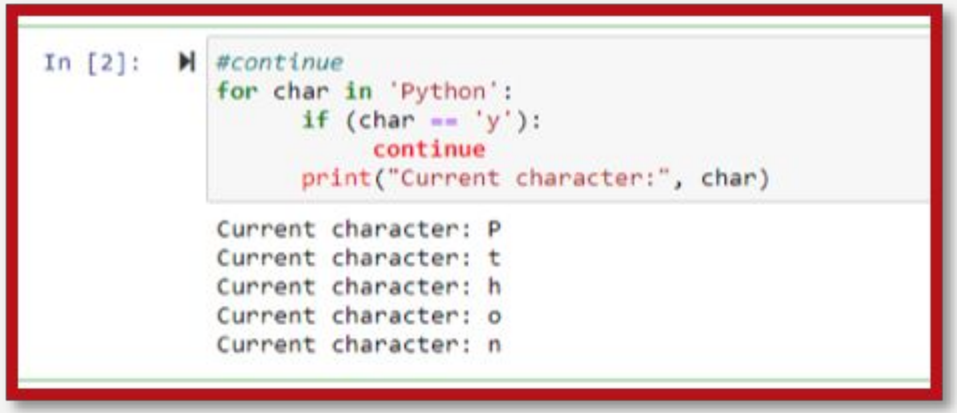
break

3. Pass Statement:

Pass statement is python is a null operation, which is used when the statement is required syntactically. It is a placeholder for future code. When the `pass` statement is executed, you avoid getting an error as empty code is not allowed in loops, function definitions, class definitions, or in if statements.

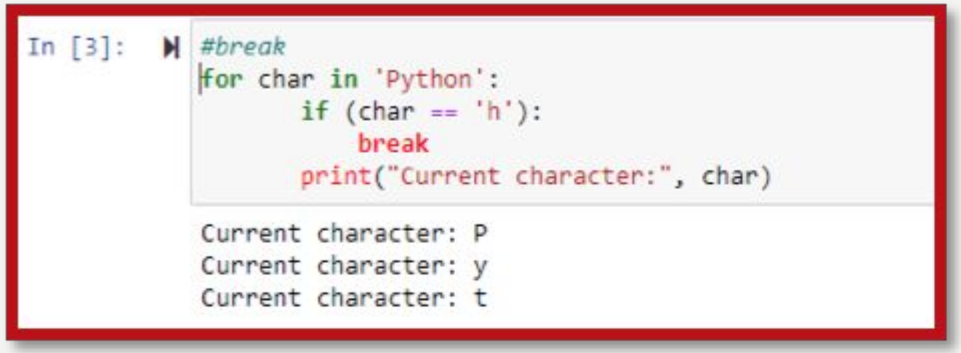
Syntax:

pass

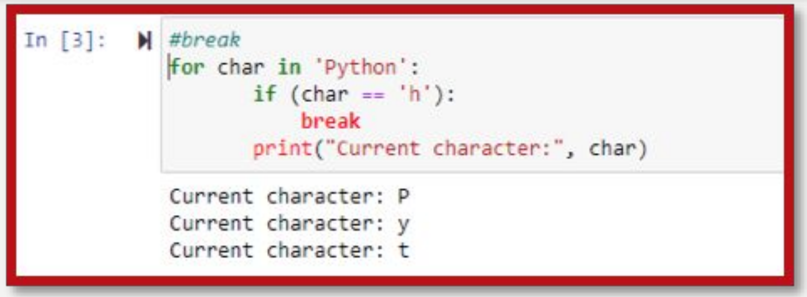


```
In [2]:  #continue
         for char in 'Python':
                 if (char == 'y'):
                     continue
                 print("Current character:", char)

         Current character: P
         Current character: t
         Current character: h
         Current character: o
         Current character: n
```

In the above example during the second iteration if the condition evaluates to true, then it will execute the continue statement. So whatever statements are present below, for loop will be skipped, hence letter ‘y’ is not printed



```
In [3]:  #break
         for char in 'Python':
                 if (char == 'h'):
                     break
                 print("Current character:", char)

         Current character: P
         Current character: y
         Current character: t
```



```
In [3]:  #break
         for char in 'Python':
                 if (char == 'h'):
                     break
                 print("Current character:", char)

         Current character: P
         Current character: y
         Current character: t
```

Functions

As our Python programs continue to get larger and more complex, we may notice that we are reusing certain pieces of code over and over again.

Functions also allow us to break our larger programs up into smaller, easier to understand steps. This makes our code much easier to read, test, and debug.

Basic structure of function

Defining function

```
def function_name():
    <block of statements>
```

Calling the function in the program

```
function_name()
```

Basic structure of function

1

function definition in Python begins with the word def, which is short for define. We define a new function using the special def keyword.

```
def function_name():
    <block of statements>
```

2

Next, we write the name of the function. Function names in Python follow the same rules as variable names:

A function name must begin with a letter or underscore _.

Function names beginning with an underscore _ have special meaning, so we won’t use those right now.

Lowercase

A function name may only include letters, numbers, and underscores _.

After the function name, we see a set of parentheses (). This is where we can add the parameters for the function.

3

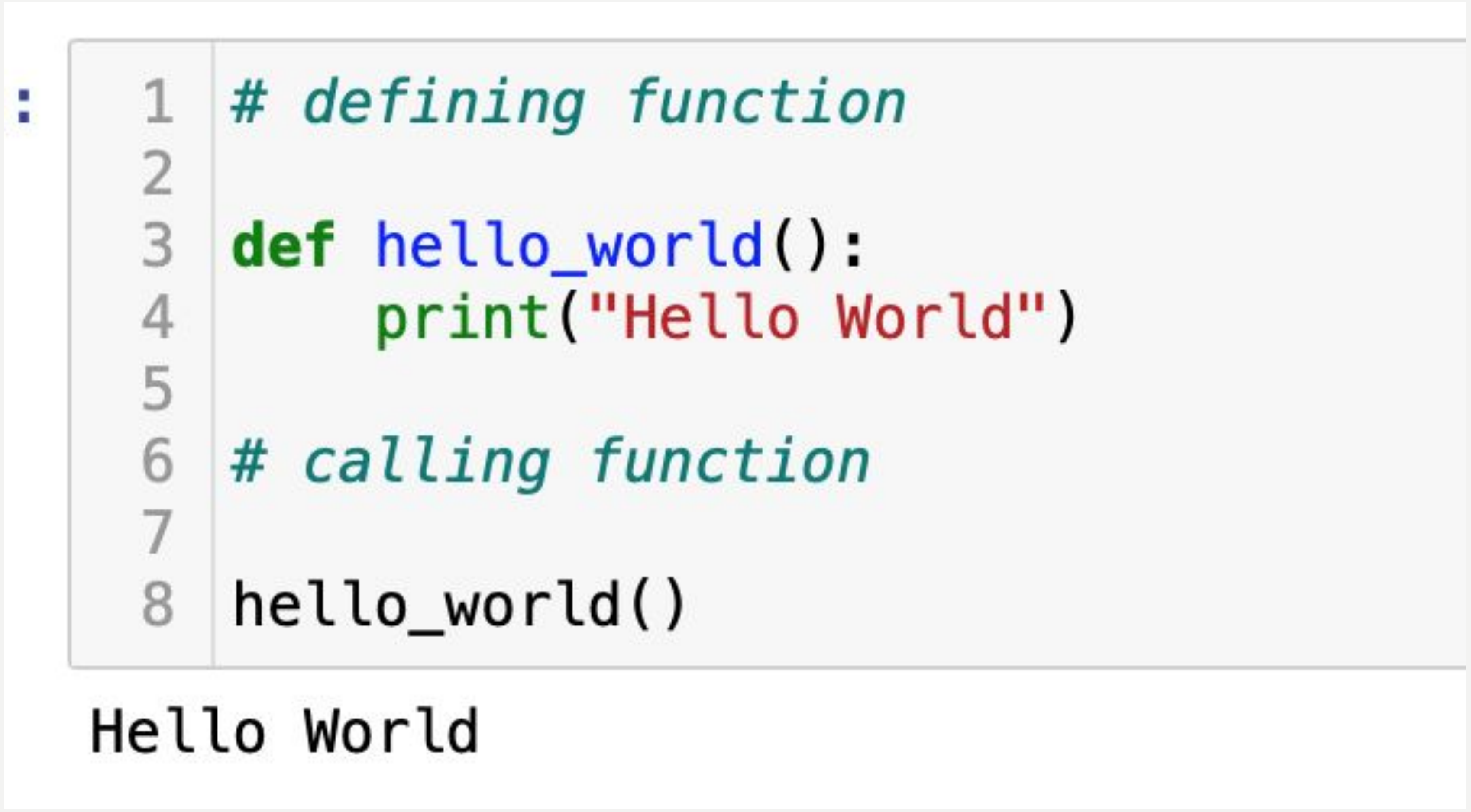
Then, we end the first line of a function definition in Python with a colon :.

This tells us that the block of statements below the function definition is what should be executed when we run the function. We’ve already seen colons in both conditional statements and loops, and here it is used in a similar way.

4

Finally, we have the block of statements that are part of the function itself. This block of statements must be indented one level to indicate that it is part of the function and not something else.

Function definition

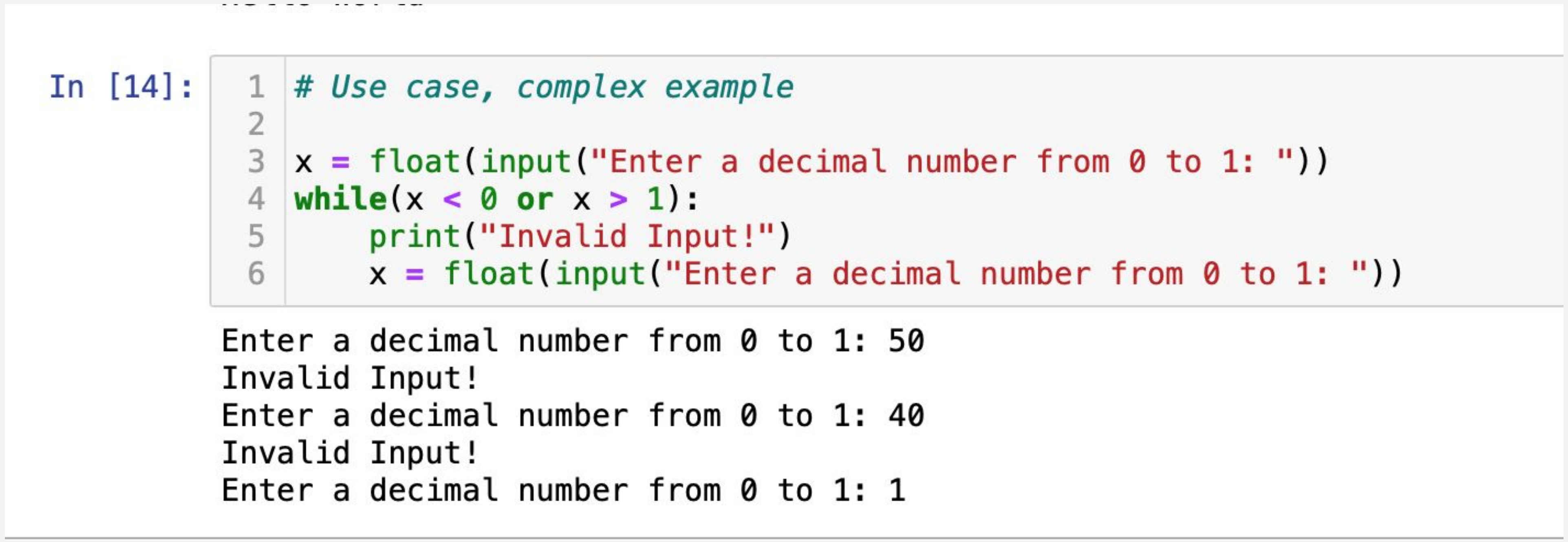


```python
# defining function

def hello_world():
    print("Hello World")

# calling function

hello_world()
```

```
Hello World
```

Function use cases

A great example is the loop structure to repeatedly get input from the user until a valid input is received.



In [14]:

```python
# Use case, complex example

x = float(input("Enter a decimal number from 0 to 1: "))
while(x < 0 or x > 1):
    print("Invalid Input!")
    x = float(input("Enter a decimal number from 0 to 1: "))
```

```
Enter a decimal number from 0 to 1: 50
Invalid Input!
Enter a decimal number from 0 to 1: 40
Invalid Input!
Enter a decimal number from 0 to 1: 1
```

Q & A